REMARKS

Claim Rejections – 35 USC § 112

Claims 1-7, 10-16 stand rejected under 35 U.S.C. 1 12, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. The examiner states, "The claims are indefinite because of lacking antecedent basis for '*said undefined operator*' in the limitation '*assigning said target type to said undefined operator*'. Furthermore, implying 'said operator' in dependent claims 2-7, 1 1-16 is indefinite."

Applicants have amended "said undefined operator" to read "said operator." Applicants have amended claim 1 to read "an expression *comprising an operator*" (amended portion added) to add antecedent basis for the term "said operator" in the dependent claims.

Claims 8-9, 17-18 and 26-27 stand rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. The examiner states, "The claims are indefinite because it is unclear 'operator' in the claims. The claims use three different terms for operator. 'an overloaded binary operator', 'parsing to determine the operator', and 'assigning said target type to said undefined operator'. It is unclear what kind of operators recited in the claims. It appears lacking antecedent basis for operator. Furthermore, claims 8, 17, and 26 recite 'all types which result from the operation of said overloaded binary operator', it lacks antecedent basis for the operation used in the claims."

For the sake of expediting prosecution, applicants have amended "parsing to determine the operator" to read "parsing to determine said overloaded binary operator." However, applicants respectfully note that, pursuant to MPEP 2173.05(e), "an overloaded binary operator" provides reasonable antecedent basis for "the operator" (*See e.g.* "'controlled stream of fluid' provided reasonable antecedent basis for 'the controlled fluid'").

Applicants have amended "assigning said target type to said undefined operator" to read "...said overloaded binary operator."

Claims 20-24 stand rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. The examiner states, "The claims are indefinite because of lacking antecedent basis for 'said operator' used in the claims."

As supra, applicants respectfully submit that the language "an undefined operator" in claim 19 provides sufficient antecedent basis for the uses of "said operator." However, in the interest of expediting prosecution, applicants have amended all instances of "an operator" or "said operator" to read "…overloaded operator."

Claim Rejections – 35 USC § 102

Claims 1-27 stand rejected under 35 U.S.C. 102(b) as being anticipated by "C# Language Specification", Version 0.28, May 2001 (hereinafter: C# Language Specification).

Applicants have amended claim 1 to include the limitations, "determining as said target type a most encompassed type from among a first set of types of loosely-typed operands…" and "~~wherein~~ at least one of said operands ~~is of an undefined~~ being a user-defined type…" These amendments are supported by the specification in at least paragraphs [0014] and [0018], respectively. Applicants respectfully submit that these limitations is not disclosed by the C# Language Specification.

C#, as disclosed in The C# Language Specification is a strongly-typed language, where variable types must be disclosed before using them. See the code sample on page 2 of the reference, which reads:

```
using System;
class Class1
{
      public int Value = 0;
}
class Test
{
      static void Main() {
      int val1 = 0;
      int val2 = val1;
```

```
        val2 = 123;
        Class1 ref1 = new Class1();
        Class1 ref2 = ref1;
        ref2.Value = 123;
        Console.WriteLine("Values: {0}, {1}", val1,
val2);
        Console.WriteLine("Refs: {0}, {1}",
ref1.Value, ref2.Value);
        }
}
```

In that code, variables `val1` and `val2` are declared to be of type `int` before they are used, while variables `ref1` and `ref2` are declared to be of type `Class1` before they are used. This is on contrast to the claimed "loosely-typed operands" of the present claim 1. A loosely typed language does not require the type of a variable to be explicitly stated, before use, or otherwise. For instance, an excerpt of the above code in a loosely-typed pseudo-code may read:

```
val1 = 0;
val2 = val1;
val2 = 123;
ref1 = new Class1();
ref2 = ref1;
ref2.Value = 123;
```

While a loosely-typed language may be easier to program than a strongly-typed language because less typing and explicit calls are required of the programmer, it may be a more difficult language to debug because fewer errors may be caught by the compiler during compilation. Further, with a loosely-typed language, a compiler must have some way of determining how to evaluate an inconsistent expression. One way to do this is by using widening conversions recited as limitations in claim 1.

Further, The C# Language Specification fails to disclose determining a target-type for an expression comprising at least one user-defined type, as claimed, *supra*. The examiner states that the language of claim 1 (before the present amendments),

> determining as said target type a most encompassed type from among a first set of types, where said first set of types comprises all resulting types of all first variant expressions, where each of said first variant expressions comprises said target expression with at least one of said operands, wherein at least one operand is of an undefined type, replaced using

> widening type conversion, where said second set of types comprises all resulting types of all second variant expressions, where each of said second variant expressions comprises said target expression with at least one of said operands, wherein at least one operand is of an undefined type, replaced using at least one of widening and narrowing type conversion; and

The examiner states that this is taught by the C# Language Specification:

> See p. 98-100, For "determining", see the definitions for most encompassed types among set of type A, and most encompassing type among set of type B (p. 99, the seventh ??, eighth ?? and ninth ??); and the discussions of *user-defined implicit conversions and user-defined explicit conversions*. These definitions/discussions have the means for determining a type as a most encompassed type from among a set of types and the means for determining a type as a most encompassing type from among a set of types. These are incorporated in the teachings disclosed in p. 107.

(emphasis added).

Applicants respectfully disagree. The cited portions teach user-defined conversions, that is the user, or programmer, defines a way to convert a variable from a first type to a second type. This is in contrast to the present claim limitations, which concern a way for the compiler to convert a variable from a user-defined type to a known type, so the user/programmer does not have to do it his or herself.

Applicants respectfully submit that independent claims 8, 10, 17 and 19 are in condition for allowance insomuch as they recite similar limitations as claim 1, and that dependent claims 2-7, 9, 11-16, 18, and 20-25 are in condition for allowance for at least the reason that the independent claims are in condition for allowance.

The examiner has rejected claims 26 and 27 under 35 USC 102 because, "Claims 26-27 is merely manipulating a mathematic algorithm." Applicants respectfully submit that this, if true, is not grounds for a rejection under 35 USC 102, and submit that claims 26 and 27 are thus in condition for allowance.

Date: December 31, 2008

/Peter Trahms-Neudorfer/
Peter Trahms-Neudorfer
Registration No. 59,282

Woodcock Washburn LLP
Cira Centre
2929 Arch Street, 12th Floor
Philadelphia, PA 19104-2891
Telephone: (215) 568-3100
Facsimile: (215) 568-3439